(12) UK Patent Application (19) GB (11) 2 262 825 (13) A

(43) Date of A publication 30.06.1993

(21) Application No 9221631.6

(22) Date of filing 15.10.1992

(30) Priority data
(31) 815331 (32) 27.12.1991 (33) US

(51) INT CL[5]
G06F 15/16 3/06

(52) UK CL (Edition L)
G4A AFL

(56) Documents cited
EP 0364115 A2 US 4775931 A

(58) Field of search
UK CL (Edition K) G4A AFGX AFL AFN
INT CL[5] G06F 3/06 9/24 13/00 15/16
Online database: WPI

(71) Applicant
Intel Corporation

(Incorporated in the USA - Delaware)

2200 Mission College Boulevard, Santa Clara, California 95052, United States of America

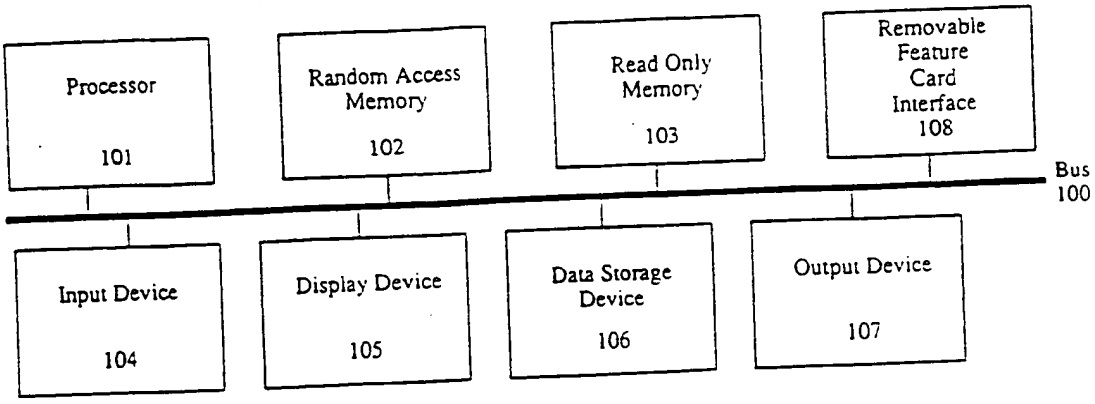
(72) Inventor
John I Garney

(74) Agent and/or Address for Service
Potts, Kerr & Co
15 Hamilton Square, Birkenhead, Merseyside, L41 6BR, United Kingdom

(54) Device driver configuration in a computer system

(57) A computer system for dynamically configuring device drivers of removable system resources. The computer system comprises a processor 101, a system memory 103, 102 and an interface 108 for receiving removable system resources such as feature cards. Each feature card includes a card memory area comprising: 1) a full device driver portion, and 2) a stub device driver portion (figure 3). Upon insertion of a card into the computer system, the device driver stub code image is read from the card memory area and transferred into an area of computer system memory. The device driver stub code is then executed by the processor of the computer system from computer system random access memory 102. Upon execution, the device driver stub enables access to the full card resident device driver by allowing memory mapping to the full device driver. The full device driver may then be activated by the processor. Upon removal of a feature card from the computer system, the device driver stub disables access to the full card resident device driver by disallowing memory mapping to the full device driver. In this way special functions like expansion memory and modems can be added to the system without the system being reset or reinitialised.

Figure 1



At least one drawing originally filed was informal and the print reproduced here is taken from a later filed formal copy.

This print takes account of replacement documents submitted after the date of filing to enable the application to comply with the formal requirements of the Patents Rules 1990.

GB 2 262 825 A

Figure 1

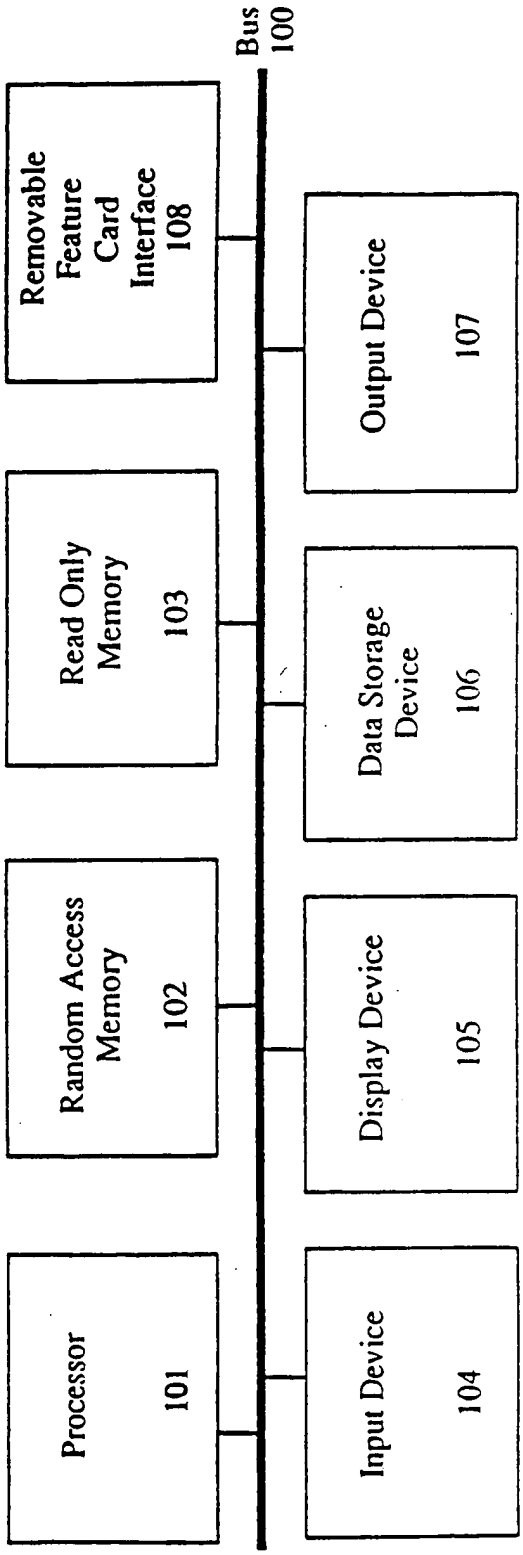

Bus
100
Removable Feature Card Interface 108
Read Only Memory 103
Random Access Memory 102
Processor 101
Output Device 107
Data Storage Device 106
Display Device 105
Input Device 104

FIGURE 2

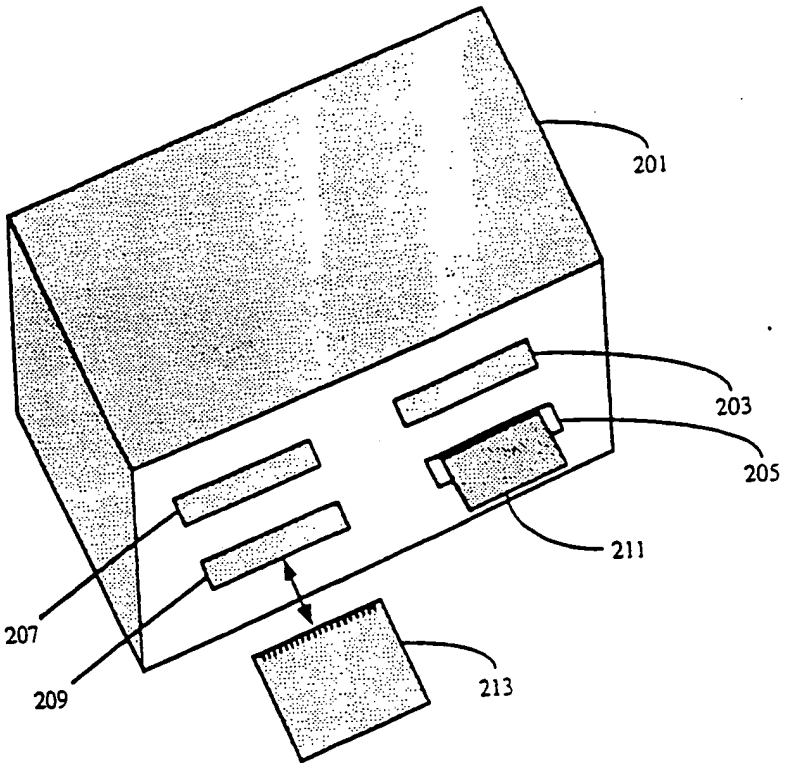


FIGURE 3

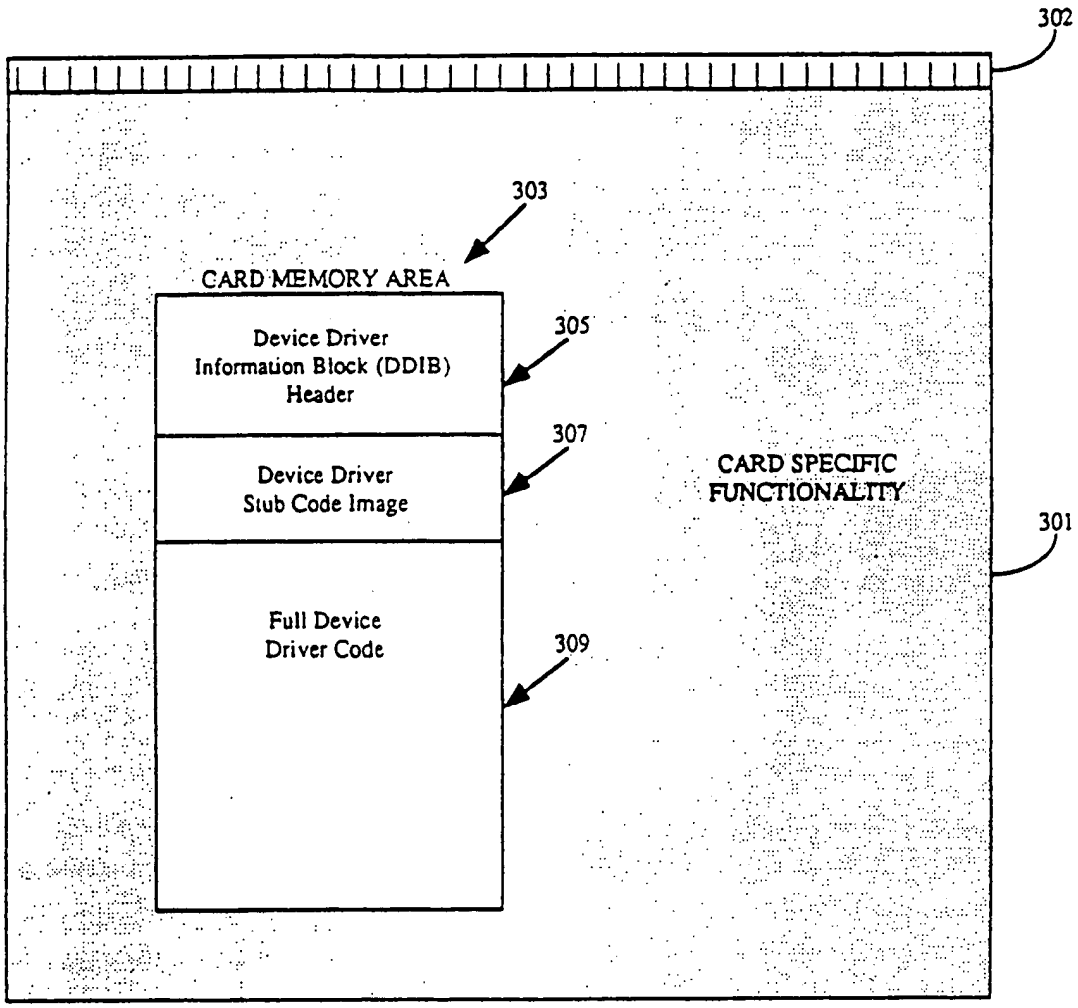
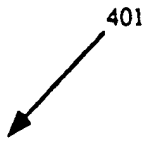


FIGURE 4



Device Driver Information Block Identity Code	403
Link Data	405
Device Driver Stub Unique Identification	407
Device Driver Linkage Information	411
Device Driver Attribute Information	413
Device Driver Strategy Offset	415
Device Driver Interrupt Offset	417
Device Driver Units and Name	419
Device Driver Stub Code Offset	421
Device Driver Stub Code Length	423
Device Driver Stub Data Offset	425
Device Driver Stub Data Length	427

Device Driver Information
Block (DDIB) Header

FIGURE 5

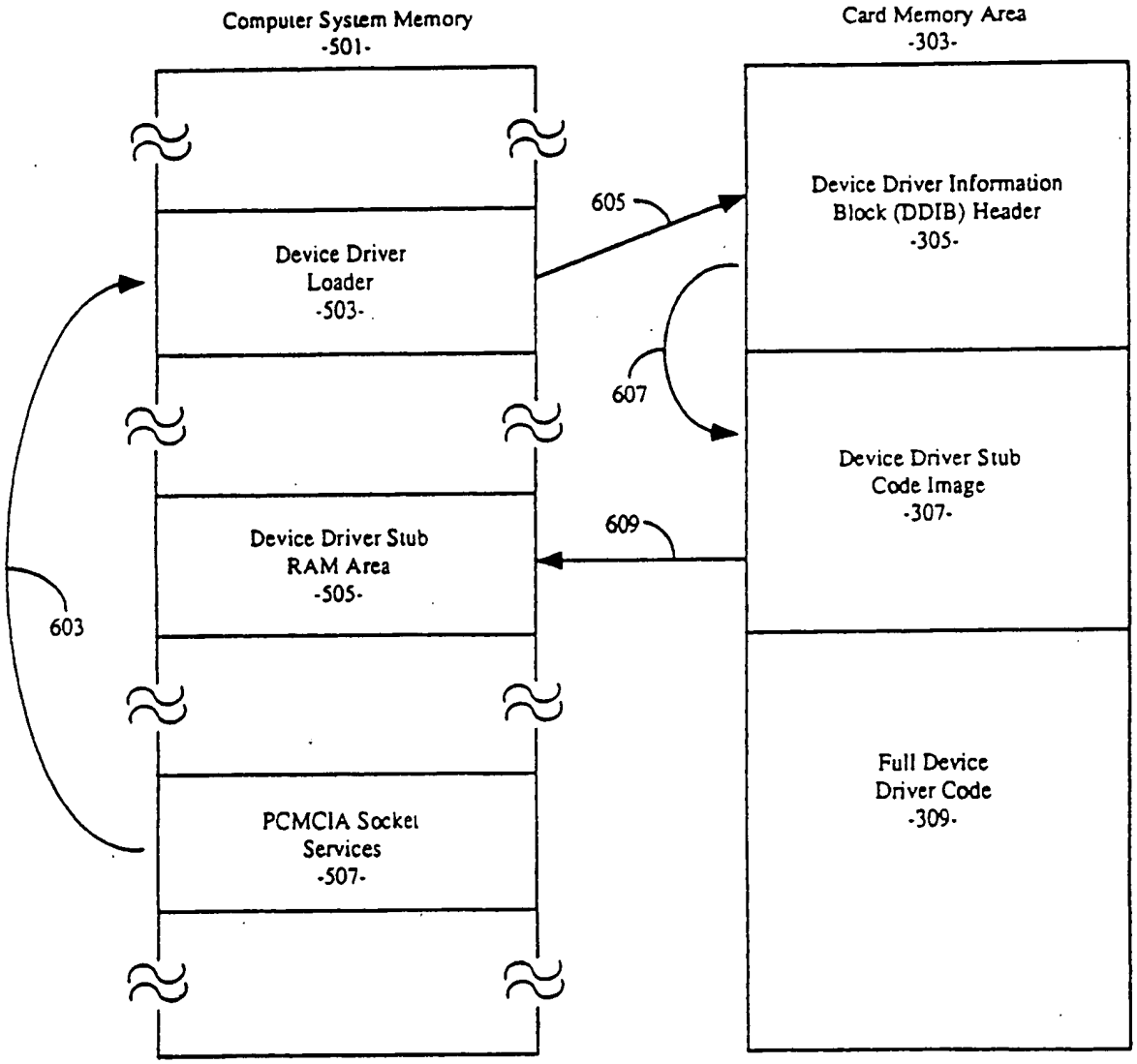


FIGURE 6c
Stub Header
-540-

Device Driver Linkage Information	-630
Device Driver Attribute Information	-632
Device Driver Strategy Offset	-634
Device Driver Interrupt Offset	-636
Device Driver Units and Name	-638

FIGURE 6d
Stub Data
-542-

Pointer to previous Stub Block	-660
Pointer to next Stub Block	-662
Adapter Identification	-664
Socket Identification	-666
Device Driver Stub Unique Identification	-668
Card Insertion Flag	-672
Driver Specific Data Area	-674

FIGURE 6c
Stub Header
-540-

Device Driver Linkage Information	-630
Device Driver Attribute Information	-632
Device Driver Strategy Offset	-634
Device Driver Interrupt Offset	-636
Device Driver Units and Name	-638

FIGURE 6d
Stub Data
-542-

Pointer to previous Stub Block	-660
Pointer to next Stub Block	-662
Adapter Identification	-664
Socket Identification	-666
Device Driver Stub Unique Identification	-668
Card Insertion Flag	-672
Driver Specific Data Area	-674

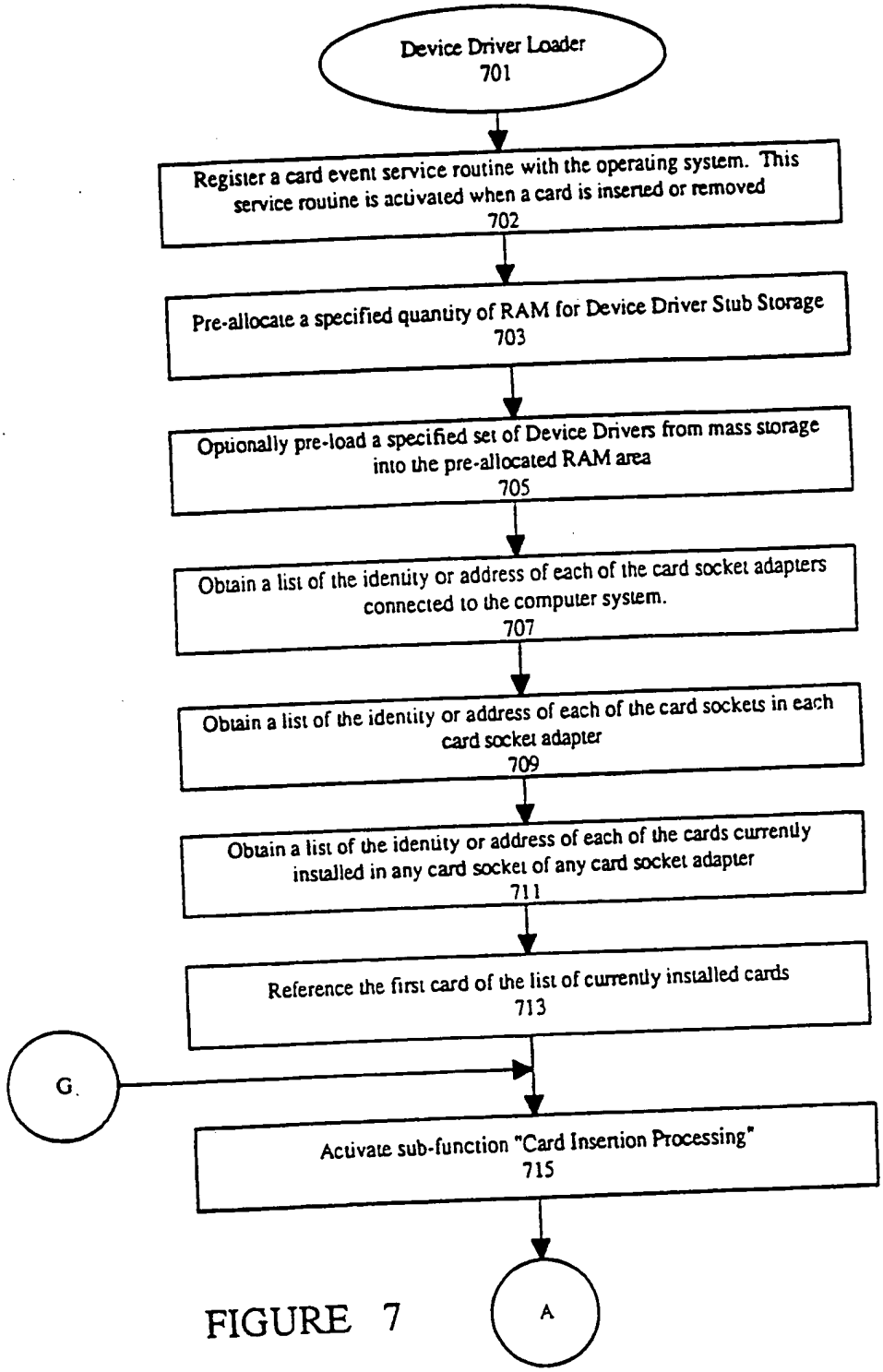

Device Driver Loader
701
Register a card event service routine with the operating system. This service routine is activated when a card is inserted or removed
702
Pre-allocate a specified quantity of RAM for Device Driver Stub Storage
703
Optionally pre-load a specified set of Device Drivers from mass storage into the pre-allocated RAM area
705
Obtain a list of the identity or address of each of the card socket adapters connected to the computer system.
707
Obtain a list of the identity or address of each of the card sockets in each card socket adapter
709
Obtain a list of the identity or address of each of the cards currently installed in any card socket of any card socket adapter
711
Reference the first card of the list of currently installed cards
713
G
Activate sub-function "Card Insertion Processing"
715
A

FIGURE 7

FIGURE 8

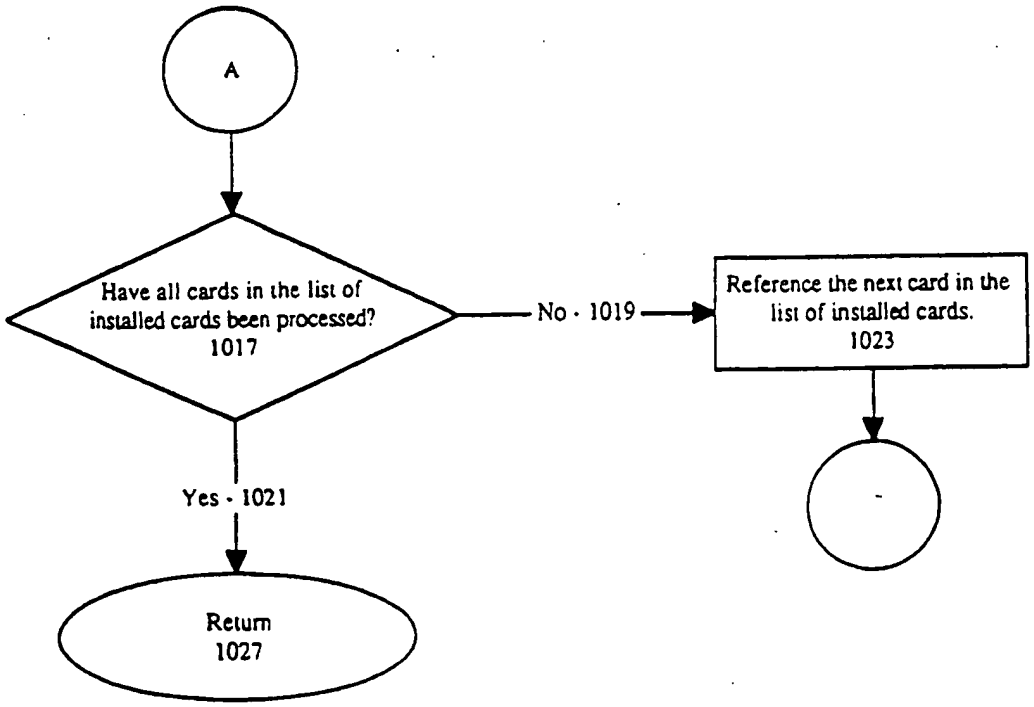

A
Have all cards in the list of installed cards been processed?
1017
No - 1019
Reference the next card in the list of installed cards.
1023
Yes - 1021
Return
1027

FIGURE 9

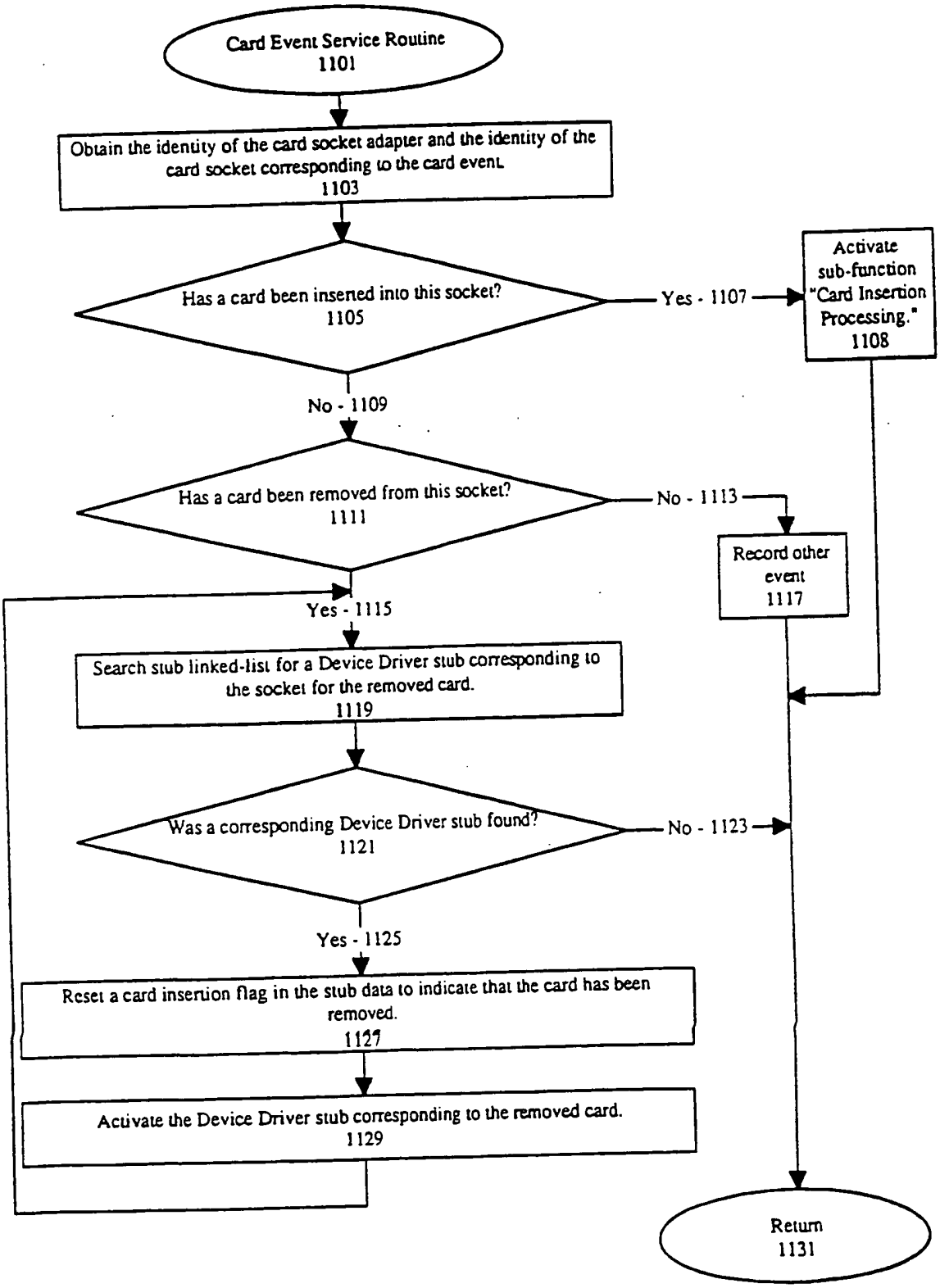


FIGURE 11

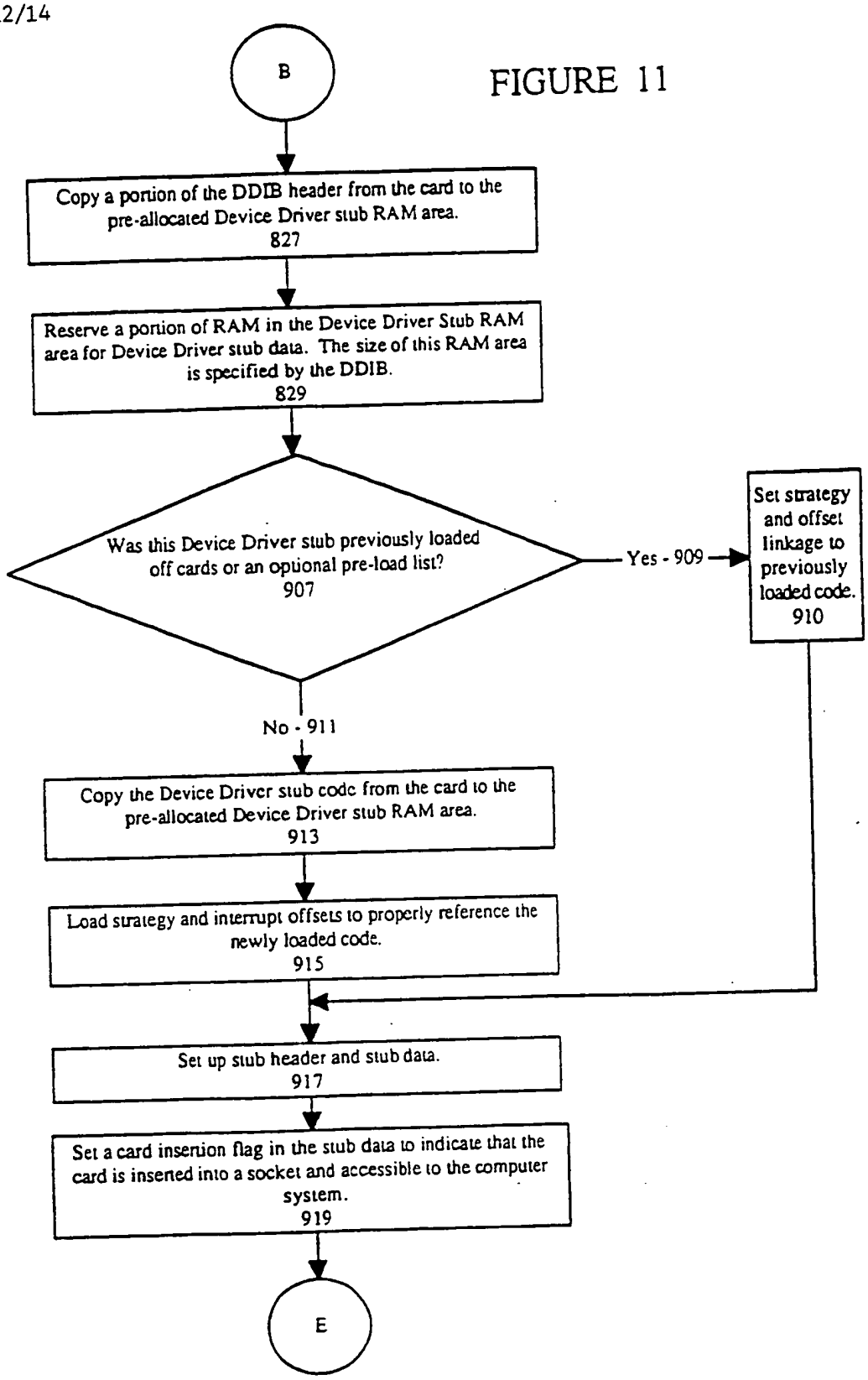
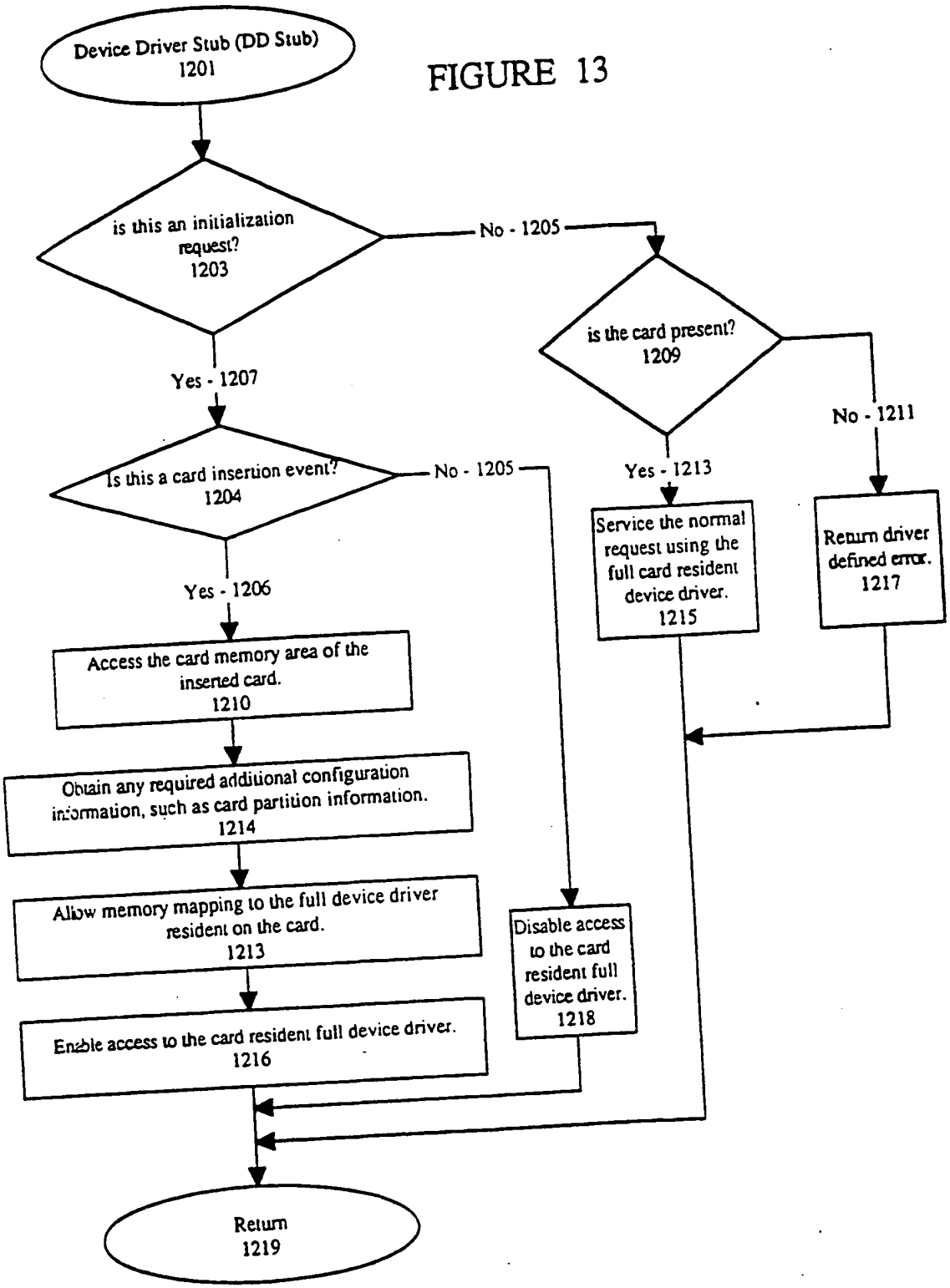


FIGURE 13


Device Driver Stub (DD Stub)
1201
is this an initialization request?
1203
No - 1205
Yes - 1207
Is this a card insertion event?
1204
No - 1205
Yes - 1206
Access the card memory area of the inserted card.
1210
Obtain any required additional configuration information, such as card partition information.
1214
Allow memory mapping to the full device driver resident on the card.
1213
Enable access to the card resident full device driver.
1216
is the card present?
1209
No - 1211
Yes - 1213
Service the normal request using the full card resident device driver.
1215
Return driver defined error.
1217
Disable access to the card resident full device driver.
1218
Return
1219

for a specific application is thereby optimized. The use of removable feature cards is particularly significant for portable computers or lap top computers where space constraints increase the need for system resource optimization. The design and use of hardware devices under the PCMCIA standard are well known in the art. It will be apparent to those skilled in the art that other implementations of removable system resources are possible.

Virtually all computer systems operate with some sort of operating system or software processing logic. The use of an operating system in a computer system is well-known in the art. The operating system is responsible for managing the processing and transfer of information between various system resources. One well known technique for managing these resources is the use of device drivers. Device drivers are software modules comprising processing logic for controlling the low level or device specific components of a particular computer system resource. For example, a device driver may be used for controlling a magnetic disk drive device coupled to a computer system. In this example, the device driver would control the various hardware specific registers, latches, signals, or other components of the magnetic disk drive device. Similarly, other computer system resources such as serial or parallel input/output (I/O) ports, modem devices, computer network interface devices, or memory expansion boards are controlled by device drivers.

In conventional computer systems, device drivers are typically loaded into random access memory (RAM) during bootstrap initialization of the computer system. Many prior art computer systems require that device drivers be loaded at initialization time in order for random access memory to be allocated properly. Depending upon the complexity of the device controlled by the device driver, the device driver itself may be relatively small or a very large device driver that consumes many thousands of bytes of random access memory. Thus, many prior art systems require that a full system configuration of resources be installed and available at bootstrap initialization time. If system resources or interfaces are subsequently added or removed from the system, the

removed from the computer system, the device driver controlling the operation of the feature card becomes inaccessible to the computer system. In most cases, the computer system requires access to a device driver in order to properly terminate the operation of the device prior to removal of the feature card. Typically, the computer system does not have sufficient time to access the device driver prior to removal of the feature card. Thus, system errors often result from an improperly terminated system resource.

Other computer systems having means for interfacing with removable electronic feature cards, provide a very limited capability for responding to insertion or removal of feature cards during post initialization operation of the computer system. Some computer systems do not recognize system resources connected to the computer system after the bootstrap initialization process has been completed. Other computer systems suspend or freeze the operation of the computer system if a system resource is removed after initialization is complete. Still other computer systems require that the system be powered down or the bootstrap initialization process be reinitiated if a new configuration of system resources is desired.

Thus, a better means for dynamically configuring system resources in a computer system is needed.

area of computer system memory. The device driver stub code is then executed by the processor of the computer system from computer system random access memory. Conversely, the full device driver code is not transferred to the computer system random access memory; rather, the full device driver is executed while still resident on the card. Upon execution, the device driver stub enables access to the full card resident device driver and allows memory mapping to the full device driver. The full device driver may then be activated by the processor.

The DDIB header comprises a set of information for linking the card device driver in a linked list with other device drivers and with the operating system logic executing within the computer system. By traversing the linked list, a particular device driver may be located. Upon insertion of a card, the linked list of device drivers is traversed to determine whether the device driver stub already resides in said computer system memory. If so, the operation of copying the device driver stub into computer system memory is prevented from occurring. As a card is inserted, a card insertion flag is set to indicate the removable system resource is coupled to the computer system.

When a card is removed from the computer system, the linked list of device driver stubs is traversed to find all device driver stubs associated with the removed card. Each associated device driver stub is executed. The device driver stub disables access to the removed card by disallowing memory mapping to the removed card. The device driver stub is unlinked from the linked list of device driver stubs and the card insertion flag is reset to indicate that the removable system resource has been decoupled from the computer system.

It is, therefore, an object of the present invention to provide a computer system in which system resources may be added or removed prior to or following bootstrap initialization of the computer system. It is a further object of the present invention to provide a computer system wherein system resources may be reconfigured without powering down the computer system. It is a further object of the present invention to provide a computer system wherein system resources may be reconfigured without

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a block diagram of the architecture of a computer system in which the present invention operates.

Figure 2 is an example of a computer housing containing a plurality of feature card insertion slots.

Figure 3 is a block diagram of the contents of a removable electronic feature card.

Figure 4 illustrates the content of the Device Driver Information Block Header.

Figure 5 illustrates the content of computer system memory as related to the content of the feature card memory.

Figure 6a illustrates the content of the Device Driver Stub RAM Area.

Figure 6b illustrates the content of a stub block.

Figure 6c illustrates the content of a stub header.

Figure 6d illustrates the content of the stub data area.

Figures 7-13 are flow charts illustrating the processing logic of the preferred embodiment.

Removable feature cards which may be removably inserted into interface 108 generally comprise electronic microcircuits within a thin housing including a detachable multiple connector interface with which the feature card may be removably inserted into a slot in a computer system housing. In the preferred embodiment, the feature cards and feature card interface 108 used with the present invention adhere to the PCMCIA release 2.0 standard for electronic feature cards. Feature cards of this form are well known to those of ordinary skill in the art.

Referring now to Figure 2, an illustration of a computer system housing having a plurality of feature card interfaces (203, 205, 207, and 209) is illustrated. As shown, feature cards 211 and 213 may be removably inserted and thereby electrically coupled to an interface 108 within the computer system. This feature card structure facilitates the convenient insertion and/or removal of feature cards during the course of a computing session.

Referring now to Figure 3, the structure of a typical feature card 301 is illustrated. Feature card 301 includes an interface 302 with which the feature card 301 may be removably electrically coupled to a computer system. Feature card 301 also includes a card memory area 303. Card memory area 303 includes software for controlling the remaining card specific functionality. This software within card memory area 303, including both data and processing logic, includes a device driver for controlling the operation of the feature card.

As described above, it is convenient to store the device driver for a feature card on the feature card itself. Using this technique, RAM space within the computer system does not need to be provided for the card device driver and processing time is not consumed in transferring a device driver from the card to computer system RAM. Maintaining the card device driver on the card itself also achieves the advantage of assuring that the device driver is always compatible with the card specific functionality. There is less chance for a mismatch between the device driver stored in the computer system and the card specific functionality provided on the card.

memory 102; rather, the full device driver is executed while still resident on card 301. Upon execution, the device driver stub enables access to the full card resident device driver 309 and allows memory mapping to the full device driver 309. The full device driver 309 may then be activated by the processor 101.

Referring now to Figure 4, the content and structure of the device driver information block (DDIB) header is illustrated. DDIB header comprises a set of information for linking the card device driver with operating system logic executing within the computer system.

DDIB header comprises a device driver information block identity code 403 that identifies the remaining information as being part of a DDIB header. Link data field 405 is used for linking the DDIB with other DDIBs (not shown) in the card memory area 303. Device driver stub unique identification 407 is a unique value that identifies the device driver stub and distinguishes the device driver stub from all other device driver stubs.

The next five DDIB header fields (i.e. fields 411, 413, 415, 417, and 419) are all the same values contained within a standard operating system device driver header. Specifically, these five parameters are contained within the DOS (Disk Operating System developed by Microsoft, Corp., Redmond Wa.) device driver header which is well known to those of ordinary skill in the art. Device driver linkage information 411, device driver attribute information 413, and device driver units and name 419 comprise device driver identification and linking information used by the operating system to identify and link with the corresponding device driver. The device driver strategy offset 415 and device driver interrupt offset 417 contain the offset from the beginning of the device driver stub code area. These fields are modified by the operation of the present invention as will be described below. Device driver stub code offset 421 and device driver stub code length 423 provide a means by which the computer system processing logic may determine where and how large the device driver stub code segment is as resident on the feature card. Similarly, device driver stub data offset 425 and device driver stub data length 427 provide a means for determining where and how large the device driver stub data

services 507 provides device driver loader 503 with an identification of the socket adapter and socket for which the card event interrupt was received. Device driver loader 503 then accesses the device driver information block (DDIB) header 305 on the newly inserted card as indicated by line 605. By accessing DDIB header 305, device driver loader 503 gains access to the card information described above in connection with Figure 4. Specifically, device driver loader 503 may read the device driver stub unique identification 407, device driver stub code offset 421, device driver stub code length 423, device driver stub data offset 425, and device driver stub data length 427. Using this information, device driver loader 503 determines where in card memory area 303 the device driver stub code image 307 resides. Once the location and size of device driver stub code image 307 is determined as indicated by line 607, device driver loader 503 copies the contents of device driver stub code image 307 from card memory area 303 into a portion of device driver stub RAM area 505 as indicated by line 609 in Figure 5. The device driver stub code image 307 is written into device driver stub RAM area 505 and linked into a linked list of device driver stubs maintained by device driver loader 503. The manner in which the device driver stubs are linked by device driver loader 503 is described in connection with Figures 6a and 6b.

Referring now to Figure 6a, the device driver stub RAM area 505 is illustrated. Device driver stub RAM area 505 comprises memory storage area for a plurality of device driver stub blocks. By way of example, Figure 6a illustrates stub 1 block 510, stub 2 block 512, stub 3 block 514, and stub n block 516. It will be apparent to those skilled in the art that any number of device driver stub blocks from zero to n may reside within device driver stub RAM area 505. It will be also apparent to those skilled in the art that the number of stub blocks within device driver stub RAM area 505 dynamically changes throughout the usage of the computer system. Thus, the device driver stubs do not need to be fixed in memory at bootstrap initialization time. It should also be noted that the relative size or number of memory locations required by each device driver stub block is relatively small in comparison to the full device driver code for controlling

device driver loader 503 for creating a forward and backward linked list of device driver stub blocks within device driver stub RAM area 505. Pointer 660 is a pointer to the previous device driver stub block in the linked list. Pointer 662 is a pointer to the next device driver stub block in the linked list. This doubly linked list structure is illustrated in Figure 6a by lines 517.

Referring to Figure 6a, device driver loader 503 contains a pointer to the first device driver stub block 510 in the linked list. The pointer 662 of stub 1 block 510 points to stub 2 block 512. Similarly, pointer 660 of stub 1 block 510 points back to device driver loader 503. In a similar manner, pointers 660 and 662 of each device driver stub block is used to point to the previous and next device driver stub in the linked list. Thus, the device driver stubs are forward and backward linked in a linked list. The last device driver stub block in the linked list (i.e., stub n block 516), points back to device driver loader 503 to complete the doubly linked list.

Referring again to Figure 6d, stub data 542 also comprises an adapter identification 664 and a socket identification 666. Adapter identification 664 and socket identification 666 uniquely identify the computer system hardware interface with which the device driver stub is associated. Device driver stub unique identification 668, which is the same identification as the feature card resident device driver stub unique identification 407 illustrated in Figure 4, uniquely identifies the device driver stub associated with the feature card. Card insertion flag 672 is used to retain an indication of whether the card associated with the device driver stub is inserted or removed. Driver specific data area 674 is a memory area allocated for use by the device driver stub for storage of its own data.

Referring now to Figures 7 through 13, flowcharts illustrating the processing logic used by the preferred embodiment are illustrated. It will be apparent to those skilled in the art that the processing logic described herein may be executed by processor 101 of the computer system.

Referring now to Figure 7, the processing logic associated with the device driver loader 701 is illustrated. Device driver loader logic 701 corresponds to device driver loader 503 illustrated in Figures 5 and 6a.

have been processed. Once all of the cards in the list of installed cards have been processed, processing path 1021 is taken to termination bubble 1027 where processing for the device driver loader terminates. If each of the installed cards in the list of installed cards have not yet been processed, processing path 1019 is taken to processing block 1023 where the index into the list of installed cards is advanced to point to the next installed card and processing continues at the bubble labelled G as illustrated in Figure 7. At the bubble labelled G, the card insertion processing subfunction is again activated for the newly indexed installed card.

Referring now to Figure 9, the processing for a card event service routine 1101 is illustrated. Card event service routine 1101 is a software routine registered with the operating system at bootstrap initialization of the computer system. Card event service routine 1101 is activated when a hardware event is detected by the computer system upon the insertion or removal of a feature card in any socket provided by the computer system. Upon activation of card event service routine 1101, the identity of the card socket adapter and the card socket corresponding to the hardware event is obtained in processing block 1103. If a card insertion event is detected, processing path 1107 is taken to processing block 1108 where the card insertion processing subfunction is activated for the newly installed card. Processing then terminates at return bubble 1131.

Referring again to decision block 1105, if a card has not been inserted into a socket, processing path 1109 is taken to decision block 1111. If a card removal event is detected, processing path 1115 is taken to processing block 1119 where the linked list of device driver stubs within device driver stub RAM area 505 is traversed in search of a device driver stub corresponding to the socket for the removed card. If a device driver stub corresponding to the removed card is found, processing path 1125 is taken to processing block 1127 where a card insertion flag in the stub data is reset to indicate that the corresponding card has been removed. Once the card insertion flag for the removed card has been reset, the device driver stub corresponding to the removed card is activated in processing block 1129. Activation of the device driver stub corresponding to the removed

on the device driver stub unique identification. If this device driver stub has been previously loaded, the device driver stub executable code does not need to be loaded again. Thus, if the stub has already been previously loaded, processing path 811 is taken to decision block 821 where available space within device driver stub RAM area 505 is checked. If there is available RAM space for the device driver stub data and the stub header, processing path 825 is taken to the bubble labelled B as illustrated in Figure 11. If, in decision block 821, there is not enough RAM space available, processing path 823 is taken to processing block 824 where an error in driver loading processing is reported. Card insertion processing then terminates through the bubble labelled C as illustrated in Figure 12.

Referring again to decision block 809, if the device driver stub for the newly installed feature card has not been previously loaded, processing path 813 is taken to decision block 815 where a test is made to determine if there is enough space in device driver stub RAM area 505 for the storage of the device driver stub executable code, the device driver stub data, and the stub header. If there is enough RAM space available, processing path 819 is taken to the bubble labelled B as illustrated in Figure 11. If, however, there is not enough RAM space available as a result of the test made in decision block 815, processing path 817 is taken to processing block 824 where an error in device driver loading processing is reported and processing terminates through the bubble labelled C as illustrated in Figure 12.

Referring now to Figure 11, card insertion processing continues at the bubble labelled B. At this point, it has been determined that sufficient space is available in device driver stub RAM area 505 for the storage of the device driver stub for the newly inserted feature card. In processing block 827, a portion of the DDIB header from the newly installed card is copied into the preallocated device driver stub RAM area. In particular, fields 411, 413, 415, 417, and 419 of the DDIB header are copied into fields 630, 632, 634, 636, and 638 of the stub header, respectively. An additional portion of RAM in the device driver stub RAM area is reserved for device driver stub data in processing block 829. The size of this stub data area is

card. Activation of the full card resident device driver code 309 is enabled and memory mapping to the newly installed card is allowed. Thus, the device driver stub provides a linkage between the computer system software and the card resident device driver and the card resident functionality.

If another device driver information block is present in the card memory area for the newly installed card, processing path 1013 is taken to the bubble labelled F as illustrated in Figure 10 where the header for the subsequent device driver information block is read in processing block 807. Because feature cards may contain more than one set of functionality, more than one device driver per card may be present. If, however, no other device driver information block is present for the newly installed card, processing path 1015 is taken to termination bubble 1017 where card insertion processing terminates.

Referring now to Figure 13, the processing logic for each device driver stub is illustrated starting at bubble 1201. The device driver stub processing logic is activated in response to a card insertion or removal event. For example, device driver stub processing logic is executed in response to the activation of the device driver stub logic in processing block 1129 illustrated in Figure 9 and in processing block 1009 illustrated in Figure 12.

If the device driver stub is being activated in order to initialize the operation of the device driver stub after being loaded, processing path 1207 is taken to decision block 1204. If the activation of the device driver stub is the result of a card insertion event, processing path 1206 is taken to processing block 1210 where the card memory area of the inserted card is accessed. Any necessary configuration or card partition information is obtained by the device driver stub in processing block 1214. Mapping to the full device driver resident on the feature card is enabled in processing block 1213. Access to the card resident full device driver is enabled in processing block 1216. As a result of enabling access to the card resident full device driver, processing control may subsequently be transferd to the full card resident device driver where feature card functionality may be

CLAIMS

1. In a computer system having a processor, a system memory, and an interface for receiving a removable system resource, a process for dynamically configuring device drivers of removable computer system resources, said process comprising the steps of:

receiving an indication that a removable system resource has been coupled to said interface, said removable system resource having a resource memory, said resource memory containing device driver stub processing logic and full device driver processing logic;

copying said device driver stub processing logic into said system memory;

executing said device driver stub processing logic from said system memory;

enabling access to said full device driver processing logic, said access being enabled by said device driver stub processing logic; and

executing said full device driver processing logic from said resource memory.

2. The process as claimed in Claim 1 wherein said resource memory further includes a device driver information block (DDIB) header, said process further including the step of copying said DDIB header into said system memory.

3. The process as claimed in Claim 2 wherein said DDIB header includes link data and a device driver stub unique identification, said process further including the step of linking said device driver stub processing logic with different device driver stub processing logic stored in said system memory.

4. The process as claimed in Claim 1 further including the steps of:

determining whether said device driver stub processing logic already resides in said system memory; and

resetting said card insertion flag when said removable system resource is decoupled from said interface.

11. In a computer system having a processor, a system memory, and an interface for receiving a removable system resource, a device for dynamically configuring device drivers of removable computer system resources, said device comprising:

means for receiving an indication that a removable system resource has been coupled to said interface, said removable system resource having a resource memory, said resource memory containing device driver stub processing logic and full device driver processing logic;

means for copying said device driver stub processing logic into said system memory;

means for executing said device driver stub processing logic from said system memory;

means for enabling access to said full device driver processing logic, said access being enabled by said device driver stub processing logic; and

means for executing said full device driver processing logic from said resource memory.

12. The device as claimed in Claim 11 wherein said resource memory further includes a device driver information block (DDIB) header, said device further including the means for copying said DDIB header into said system memory.

13. The device as claimed in Claim 12 wherein said DDIB header includes link data and a device driver stub unique identification, said device further including the means for linking said device driver stub processing logic with different device driver stub processing logic stored in said system memory.

14. The device as claimed in Claim 11 further including:

20. The device as claimed in Claim 18 further including:

a card insertion flag in said system memory; and

means for resetting said card insertion flag when said removable system resource is decoupled from said interface.

21. In a computer system having a processor, a system memory, and an interface for receiving a removable system resource, a process for dynamically configuring device drivers of removable computer system resources, substantially as hereinbefore described with reference to the accompanying drawings.

22. In a computer system having a processor, a system memory, and an interface for receiving a removable system resource, a device for dynamically configuring device drivers of removable computer system resources, substantially as hereinbefore described with reference to the ccompanying drawings.

Category	Identity of document and relevant passages	Relevant to claim(s)

Categories of documents

X: Document indicating lack of novelty or of inventive step.

Y: Document indicating lack of inventive step if combined with one or more other documents of the same category.

A: Document indicating technological background and/or state of the art.

P: Document published on or after the declared priority date but before the filing date of the present application.

E: Patent document published on or after, but with priority date earlier than, the filing date of the present application.

&: Member of the same patent family, corresponding document.

Databases: The UK Patent Office database comprises classified collections of GB, EP, WO and US patent specifications as outlined periodically in the Official Journal (Patents). The on-line databases considered for search are also listed periodically in the Official Journal (Patents).

Published 1993 at The Patent Office, Concept House, Cardiff Road, Newport, Gwent NP9 1RH. Further copies may be obtained from Sales Branch, Unit 6, Nine Mile Point, Cwmfelinfach, Cross Keys, Newport, NP1 7HZ. Printed by Multiplex techniques ltd, St Mary Cray, Kent.